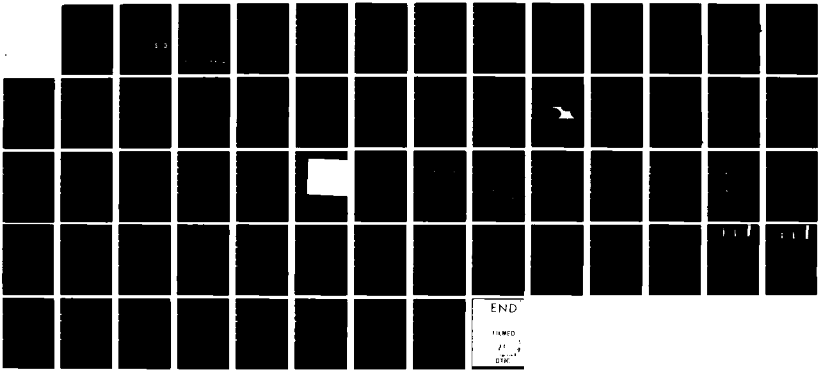
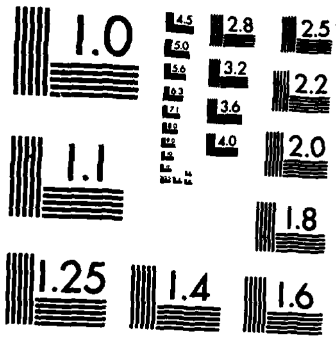
AD-A126 025 AN ARCHITECTURAL REFERENCE MODEL FOR DISTRIBUTED DBMS/FILE SYSTEM ENVIRONMENTS(U) BGS SYSTEMS INC WALTHAM MA P S MAGER ET AL. SEP 80 TR-80-010 1/1

UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(3)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER TR-80-010	2. GOVT ACCESSION NO	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Architectural Reference Model for Distributed DBMS/File System Environments		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER TR-80-010
7. AUTHOR(s) Peter S. Mager, Robert P. Goldberg		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS BGS Systems, Inc. Waltham, MA 02154		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS NUSC - New London Laboratory New London, CT		12. REPORT DATE September, 1980
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Systems, Distributed Data Bases, Data Base Management, File Systems Architecture

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents a conceptual model for distributed data sharing systems based on the client-server concept. It shows how the model can be used to design a layered set of protocols for such systems and gives an overview of the functionality required.

An Architectural Reference Model

for

Distributed DBMS/File System Environments

Prepared by:

Peter S. Mager

Robert P. Goldberg

BGS Systems, Inc.

470 Totten Pond Rd.

Waltham, Mass.

TR-80-010

September 1980

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER TR-80-010	2. GOVT ACCESSION NO.
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Architectural Reference Model for Distributed DBMS/File System Environments	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER TR-80-010
7. AUTHOR(s) Peter S. Mager Robert P. Goldberg	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS BGS Systems, Inc. Waltham, MA 02154	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS NUSC - New London Laboratory New London, CT	12. REPORT DATE September, 1980
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Systems, Distributed Data Bases, Data Base Management, File Systems Architecture

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents a conceptual model for distributed data sharing systems based on the client-server concept. It shows how the model can be used to design a layered set of protocols for such systems and gives an overview of the functionality required.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Table of Contents

List of Figures

List of Tables

1 Introduction

This report describes an architectural reference model that can be used to decompose distributed data sharing systems into component parts. The purpose is to formalize interfaces so as to facilitate the placement of components on different nodes of a distributed system. It does this by describing data access functionality in terms of two dimensions:

- a data abstraction dimension that involves a step by step transformation between the (physical) stored view of data and the (end user) displayed logical view of data
- a spatial dimension that involves the interaction between transactions and between pieces of data on different nodes.

The data abstraction dimension is based on a hierarchical model that describes the step by step transformation of data between linear bit streams that are stored on auxilliary storage media and formatted displays (records, reports, screens, etc.) that contain the view of the data presented to end users. The intermediate levels in the transformation are known as layers of data abstraction and represent the view of the data seen by some intermediate component of the system. The portion of the system that is concerned with these data transformations consists of components that take data at one level of abstraction as input and after performing appropriate data selection, manipulation and combination output that data at a different level of abstraction. Each level has its own conceptual version of a data directory or dictionary that describes how to find data objects at that level and may contain other information as well.

The spatial dimension provides a basis for partitioning the software that controls shared access to data from concurrently executing programs, terminals and job streams that may be running on different nodes of the system. That software is responsible for ensuring consistency among copies of data at different nodes (e.g. ensuring the consistency and integrity of the database at any given level of abstraction). This spatial dimension makes use of a network protocol layering to ship data between nodes. The

data base control portion of this layering is the level primarily responsible for the functions needed to ensure the integrity and consistency of distributed data management functions and is thus the level that is of primary concern in this report.

The interaction between these two layerings leads to a client-server or client-global data manager-server model of distributed data sharing that provides a framework ("formal reference model") for describing existing systems, constructing new ones, and, in general, for discussing and analyzing the functional issues and implementation tradeoffs common to such systems.

This type of model is useful for

- increasing the transportability of components between different systems and subsystems
- the design of supersystems that connect DBMSs together and allow them to share data and access each other's data
- facilitating interfacing between subsystems based on different data models
- paving the way for the design of more reliable systems by establishing a framework for efficiently supporting the redundant storage of data within fault tolerant systems.

A model of this type may eventually be the basis for interface standards that would significantly increase the transportability and reusability of software in DOD environments.

Such concerns are particularly important for the Navy since its software is characterized by

- long lead times for development
- high maintenance costs
- unusually high costs for interfacing between different subsystems and components.

These effects are magnified by the need for rapid deployment of new technologies and in particular of those supporting the increased

availability, reliability and responsiveness of computer systems.

This report presents some concepts that seem to have potential for improving the design, maintenance and general understanding of distributed data sharing systems. These include:

- the use of a hierarchical layering of data abstraction to characterize approaches to data sharing in a data model independent way. This makes it possible to isolate considerations of data modeling from issues related to data sharing and discuss coordination mechanisms independently of the level of abstraction and granularity of the data objects being shared.

- the use of logical data units (LDUs) to define the interface between modules at different nodes. These LDUs are the basis for standardizing interfaces between database modules in different computers. By parameterizing the choice of LDU we make most (possibly all) of the issues related to sharing of data independent of the format and meaning of the data being shared. Filling in the choice of LDU allows our model to be applied to different systems.

- the development of a client server model for division of functionality between front-end and back-end parts of a distributed data sharing system. This type of model is useful for very simple systems and may be most applicable to a local area network environment that ties together small computers each of which has very limited resources or to a two computer system where functionality is divided between the two computers in a cleanly interfaced way.

- the development of a client-global data manager-server model to describe networks allowing coordination between heterogeneous data base management systems at different nodes. This is a more general model that subsumes the client - server (front-end / back-end) model and provides additional functionality to support redundant copies of data and other features that characterize true distributed data sharing. These are probably needed to provide the fault tolerance and high availability that seem desirable for future Navy real time systems.

The formal reference model presented here divides global data management into three functional areas:

- those that support intra-transaction consistency, e.g. the reliable execution of a single transaction

- those that support inter-transaction consistency, e.g. the

reliability of the database in the presence of concurrent transactions that could potentially interfere with each other

- various supportive features that could be interpreted as data sharing oriented enhancements to the network environment. These might include a global directory system, various primitives to support resynchronization, and facilities to detect and recover from network problems such as the failure of an individual host.

In the course of describing the model, several inter-transaction consistency control techniques that may be particularly relevant to Navy real time systems are presented. These include:

- generalized time stamps that can be used to explicitly integrate priority considerations into the synchronization scheme
- multiple version approaches (based on time stamp ordering) that may allow faster responses to transactions under some circumstances.

Similar ideas have been used successfully in other data base related contexts, but their use together in a formal reference model is new to this report. In particular, the data abstraction and spatial layerings presented here may form the basis for construction of new types of systems that allow distributed data sharing even in contexts where several different data models are used. It may also help facilitate the design of modular software with increased portability of components between different subsystems.

Chapter 2 presents a client/server approach to dividing data access functionality into front-end and back-end parts. In this framework the client machine or front-end performs application dependent functions including handling the interface(s) to users. The server machine acts as a back-end data manager and interfaces to/controls disk storage and other mass storage devices. The chapter then goes on to describe a more general model that divides system functionality into client, server, and global data manager parts. This has the advantage of separating those parts of a system that regulate the interaction between processes at different nodes (a function that involves complex design tradeoffs) from those involving only actions associated with a single node. The latter can generally be handled in a much more straightforward manner. The rationale for a global

data manager is then explained by starting with a simple front-end/back-end system involving two nodes and introducing first multiple back-ends (servers) and then multiple front-ends (clients) to show where the need for coordination functionality comes from.

Chapter 3 introduces a data base architecture that divides database/file system functionality into eight hierarchical levels of data abstraction, each of which provides a consistent view of the data stored and used within the network environment. This layering is valid for both single node (standard centralized) and distributed DBMSs, but the layers are chosen so as to be particularly suitable for distributed systems. The chapter then goes on to explain how the architecture can be broken into client (front-end) and server (back-end) parts by choosing an appropriate layer at which to divide the functionality of the system. Lower levels of abstraction (those closer to the physical storage format) are handled by a back-end server data handler that can be anything from a disk or memory controller to a data base machine or mainframe computer. Higher levels of abstraction (those closer to the end user view of data and maintaining more interpretive information) are handled by a front-end user oriented client machine that runs application programs and some higher level transformation, manipulation and control functions related to data management. The choice of level of abstraction for the division between client and server machines is an important design parameter that greatly affects the appearance and performance of the resulting system. Several of the architectures described in [Mager 80] are then characterized in terms of this model by way of example of how it can be applied to real systems.

Chapter 4 presents a decomposition of distributed data sharing architecture into functional components. This decomposition describes a prototypical data sharing system in terms of functional modules.

These modules are divided into categories encompassing

- client (user-interface) functions
- server(data handler) function
- migratory functions that can be in either the client or server
- global data manager functions needed for coordination between

modules on different machines.

Some alternative strategies for distribution and control of global data manager functions are then described.

Chapter 5 summarizes our results, reviews the applicability of these results to Navy real time systems and suggests some areas for further research.

This report assumes familiarity with basic database concepts. A reader not familiar with these might find it useful to consult an elementary text book on database management such as [Date 75] or [Martin 77]. Chapter 2 is probably the easiest to read and should be comprehensible to an educated reader who has a moderate familiarity with basic computer concepts. Chapter 3 is more specialized and is probably most meaningful to readers familiar with at least one database management system or at least with general database management concepts. Chapter 4 is mainly of interest to designers, implementers, and planners of distributed database and file systems.

2 A Conceptual Model for Distributed Data Sharing

2.1 Overview of Chapter

One of the main goals of this report is to show how a formal reference model for distributed data sharing can be derived from a hierarchical abstract layered model for a single node system by choosing an appropriate abstract level and using that as an interface between front-end (client) and back-end (server) parts of the system. This chapter introduces the division between client and server parts and gives some examples of how a model based on this division can be applied to existing systems. It is hoped that after seeing the formal reference model represented in these simple terms, the reader will be prepared for the greater complexity that will be introduced in succeeding chapters.

This chapter describes distributed data sharing in terms of a client-server model. It then extends the model by adding a Global Data Manager (GDM) to handle potential conflicts that may arise if multiple clients and multiple servers are allowed in the system.

Each front-end or client part contains those services associated with higher levels of data abstraction - in particular those services associated with application (client or customer) functions. The back-end or server part contains those services associated with lower levels of data abstraction - in particular those functions concerned with management of data stored on permanent, usually bulk store, mass memory devices. In a sense, the server parts of the system can be thought of as being secondary service processors (servers) for the applications (clients) running on the front-end (client) parts of the system.

This chapter will give an overview of two ways of looking at this division of functionality. It will do this first in the context of a client accessing a single back-end server at a time with no interactions with other clients. It will then generalize this by allowing arbitrary interactions between clients and servers. The last part of the chapter discusses the additional functional features needed to support this and in

a sense gives a derivation of what a global data manager that handles this interaction should consist of.

The terminology used in this chapter and throughout the report has a sound historical basis. The use of the term back-end is derived from the concept of back-end data managers (and data base machines) that was originally introduced by the XDMS project at Bell Labs in the early 1970s [Canaday 74]. Originally back-end data managers were conceived of as a way of offloading work from a large host analogously to the way front-end communication processors had previously offloaded some of the work of interfacing to terminals from those hosts.

Our use of the term front-end is more general than current usage in that we use it to refer to all client (application and high level) functions not just to the communication interfacing functions handled by conventional physical communication front-ends such as IBM 3705s or COMTEN processors.

The idea of client and server machines was introduced at Xerox PARC in the mid-1970s as a way of describing the interaction between ALTO personal computers (clients) and various resource managers (servers) that interact with each other in the PARC Ethernet. Our use of these terms is more general in that we use them to describe logical functions rather than physical machines. These logical functions can be on the same or different physical machines. This separation of logical concepts from physical entities is important because it allows the reference model we are developing to be more easily generalizable to a many-client - many server environment. It also makes the choice of physical location of the server transparent to the client (and vice versa). This allows access to server functions on the same node as the client and to server functions on a different node to be handled in a uniform manner.

2.2 A Client-Server Model

Figure 2-1 shows a model of a data sharing network based on a simple client (application) - server (data handler) model. Application programs and the high level part of the data sharing system run in client machines. These transform user requests (transactions) into network messages to data handler or server machines. Each server is responsible for controlling accesses to and maintaining the integrity of its portion of the database. The unit of data transfer between client and server machines can be high level data base records, storage oriented file records, pages in a linear address space, or similar units. The choice depends on how the functionality of the system is divided between client and server.

The Xerox PARC file server systems [Swinehart 79] [Israel 78] [Paxton 79] are based on this type of approach. In their case single-user client computers (Altos) are connected to server data handlers using a high bandwidth contention-based bus (the Ethernet). The unit of data transfer is a page or block of a linear address space.

An analogous interface based on a very different hardware configuration is used by the bibliographic search system at OCLC in Ohio [Blake 79]. In their case the data handler server is a multiple (10) processor Tandem system which is internally tied together by a high speed bus. Four SIGMA IX computer systems currently act as client machines processing the application parts of user transactions. The client and server parts of the system are joined by communication links. In this case the unit of data transfer is the database records maintained by Tandem's data handler (ENSCRIBE).

A similar interface could be used to allow access from ordinary host client machines to back-end data manager servers such as the database machines being developed by the University of Ohio and Univac [Banerjee 78] [Hsiao 79] [Banerjee 79] or the multi-level data hierarchies proposed by Lam and Madnick [Lam 78] [Madnick 75] [Madnick 79] and Berra [Berra 79]. The data interface could be "database calls" (DL/1 calls in IBM's IMS system) and retrieved records, logical fragments consisting of contiguously stored parts of rows of relations, arbitrary sized blocks or pages from linear memory or other convenient data units.

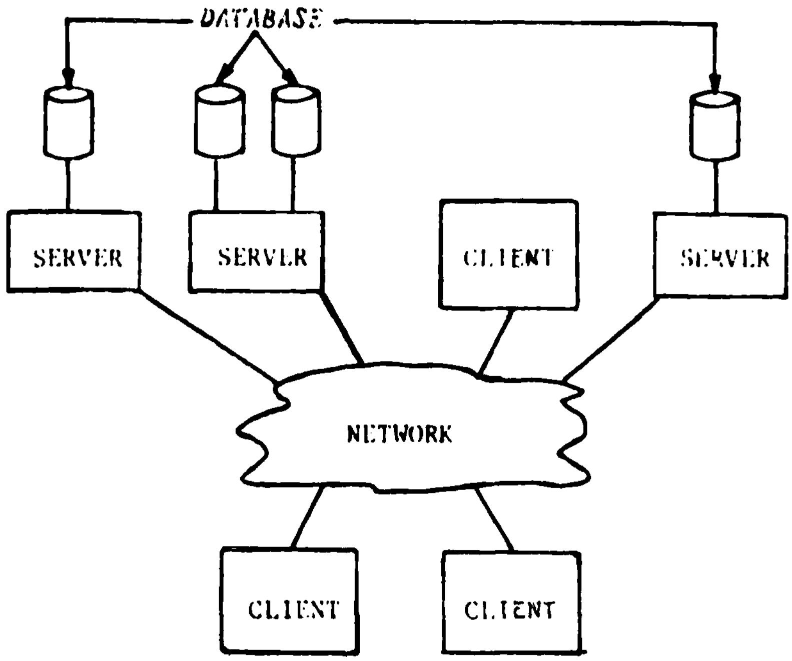


Figure 2-1: The Client Server Model for Front-End Back-End Interaction

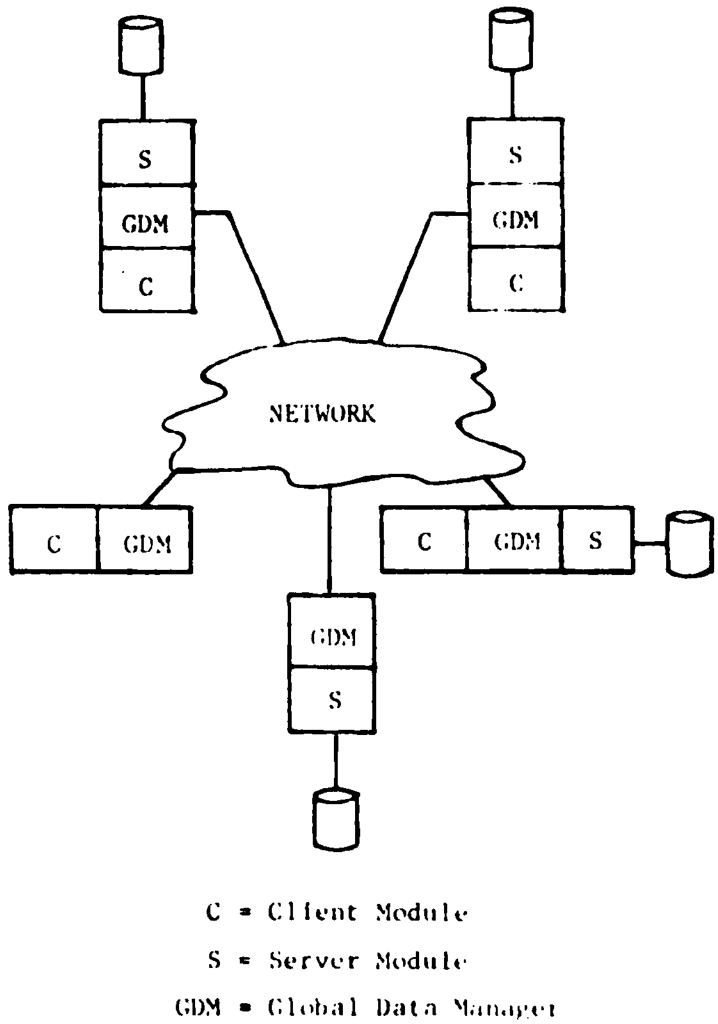


Figure 2-2: A Client-Server-GDM Model for Distributed Data Sharing

2.3 A Clients-Servers-GDM Model

A more general view of distributed data sharing is shown in Figure 2-2. In that figure both the application client (C) and data handler server (S) parts of the data sharing system can reside in the same or different nodes. All interfaces between them are controlled by a Global Data Manager(GDM) which routes data requests to the appropriate server. In this way data accesses are made independent of the server module that controls the data. The GDM is responsible for whatever consistency control and fault tolerance support are needed to make the system function reliably. This model is more general than that shown in Figure 2-1 in that some nodes may contain only client modules and a GDM interface, some may contain only a server module and the corresponding piece of the GDM, and others may contain all three types of modules. It appears that this model is more appropriate for large processors and correspondingly more complex data sharing systems since the need to have both client and server modules in the same node is more important in large multi-programming systems than in small single-user systems. A further generalization of the model shown in Figure 2-2 would allow multiple client modules (perhaps one per application program) and multiple servers at a single node. This may be useful in certain environments.

The distributed version of Cullinane's CDMS system [Cullinane 79] is based on this second type of approach. In that system IDMS corresponds to the server data manager, an application program with a small database interface corresponds to the client module, and an intermediate facility called Multiple Computer Support that runs in the same machine as client modules functions as the global data manager. Several client modules may run on the same node and interface to the servers through the same multiple computer support facility. The unit of data transfer is a data packet that includes information from both the DML (Data Manipulation Language) call and the subschema.

A similar modular division is used by CCA's SDD-1 [Rothnie 79]. In this case, the server modules are Datacomputers, the client modules are transaction managers associated with users or application programs and the Global Data Manager is an extended facility that includes reliability

enhancements to the underlying network environment. The unit of data transfer is a logical data fragment that corresponds to the part of the row of a relation that is stored contiguously.

The University of California at Berkeley's Distributed INGRES and its associated local area network variant MUFFIN [Stonebraker 79] uses a modular breakdown similar to CDMS and SDD-1. One or more master INGRESes per system contain the client-module/user- interface functionality as well as most of the functionality of a Global Data Manager. Slave INGRESes on the same or other machines function as back-end data handlers or servers. Like SDD-1, Distributed INGRES uses logical data fragments as its client-server data interface.

The generalization from a client server system, in which a server is (at least temporarily) dedicated to a single client, to a clients-servers-GDM system, in which clients request that actions be performed by a service facility consisting of multiple servers, introduces a set of requirements that form the crux of distributed data sharing functionality. It is this ability to treat a group of back-end servers as a single service facility that distinguishes distributed data sharing systems from back-end storage networks such as XDMS [Canaday 74] or the Octopus network [Watson 80] at Lawrence Livermore Labs. The functionality to support this integration of clients and servers into a closely interacting group of processes is embedded in the global data manager (GDM). To illustrate this better, it is convenient to subdivide the functional requirements for global data management in distributed systems into three main parts:

- those that allow a client machine to treat a group of back-end servers as a single service facility,
- those that allow a group of back-end servers to handle potentially conflicting requests from multiple clients, and
- those that provide enhancements to the reliability of the underlying communication network.

In our model the first type of function will be grouped together as intra-transaction consistency control functions, the second as inter-transaction consistency control functions, and the third as network

enhancements.

The first group of functions, those that support the integration of back-end servers into a single service facility, requires the GDM to support the following functions:

1. locating on which back-end server a given piece of data (data fragment) is stored,

2. choosing the "nearest" or "most easily accessible" copy of the data fragment if it is stored redundantly on multiple machines,

3. planning in which order (and with what amount of parallelism) to access data and how to combine data from different machines,

4. handling any format conversions or data translation needed for compatibility among data items stored on different machines (Translation may be required for data codes, formats and/or structural information.),

5. providing a backout mechanism to restore the data base to a consistent state if the client front-end or network crashes after performing updates on some but not all of the back-end server machines,

6. providing a mechanism to queue updates to servers that are temporarily unavailable (either because the server is down or because it is separated from the client due to network failure).

These requirements are illustrated in Figure 2-3. In the figure a single user attached to a client machine performs an update or query that accesses data stored on various machines distributed around a network. The global data manager component on this user's local machine intercepts his request and translates it into a set of messages to the back-end data managers (servers) on the nodes where the data to be retrieved or updated is stored. To do this the global data manager makes use of three principal subcomponents:

- a directory subsystem which helps locate where copies of data to be accessed are located,

- an access strategy planner which determines which copy of data to access,

- an intra-transaction consistency control function which provides reliability support and in particular ensures that redundant

copies of a data item are kept consistent.

In addition, a data format translator may be provided to perform any data conversions that are required. However, we shall see in Chapter 4 that this is more conveniently interpreted as part of the data access function rather than as a GDM activity.

Requirements (1) and (2) from our list are usually handled by the directory system, requirement (3) by the access planning function, (4) by the data translation function, and (5) and (6) by the intratransaction consistency control function and extensions to the network services. This will be discussed in more detail in chapter 4.

Allowing multiple client machines to access the server machines concurrently creates a need for a mechanism to maintain consistency among requests (especially updates) coming from different machines. The usual requirement is that a collection of transactions be executed in such a way that the concurrent execution of the group of transactions is equivalent to executing the same group of transactions serially (one at a time in a noninterleaved way) in some order [Rothnie 77] [Bernstein 79]. This requirement is relatively easy to satisfy if there is only one server machine. All that is required is a global data manager or mediator that single threads the transactions similarly to how a centralized DBMS handles transactions from terminals. (The problem becomes more complex if multi-threading of transactions involving multiple operations is allowed.) Satisfying the serializability requirement in a distributed network with multiple servers in an efficient manner is one of the more difficult tasks of distributed data base management. In addition to determining potential overlaps among data items that are accessed or manipulated by different transactions, there are problems of synchronizing operations in different machines. Both types of problems introduce functional requirements that require support facilities to be added to a standard network architecture.

The most important of these facilities that seem necessary for network synchronization are

- a time stamp mechanism or some other means to indicate the start time of transactions on a network wide basis (transaction

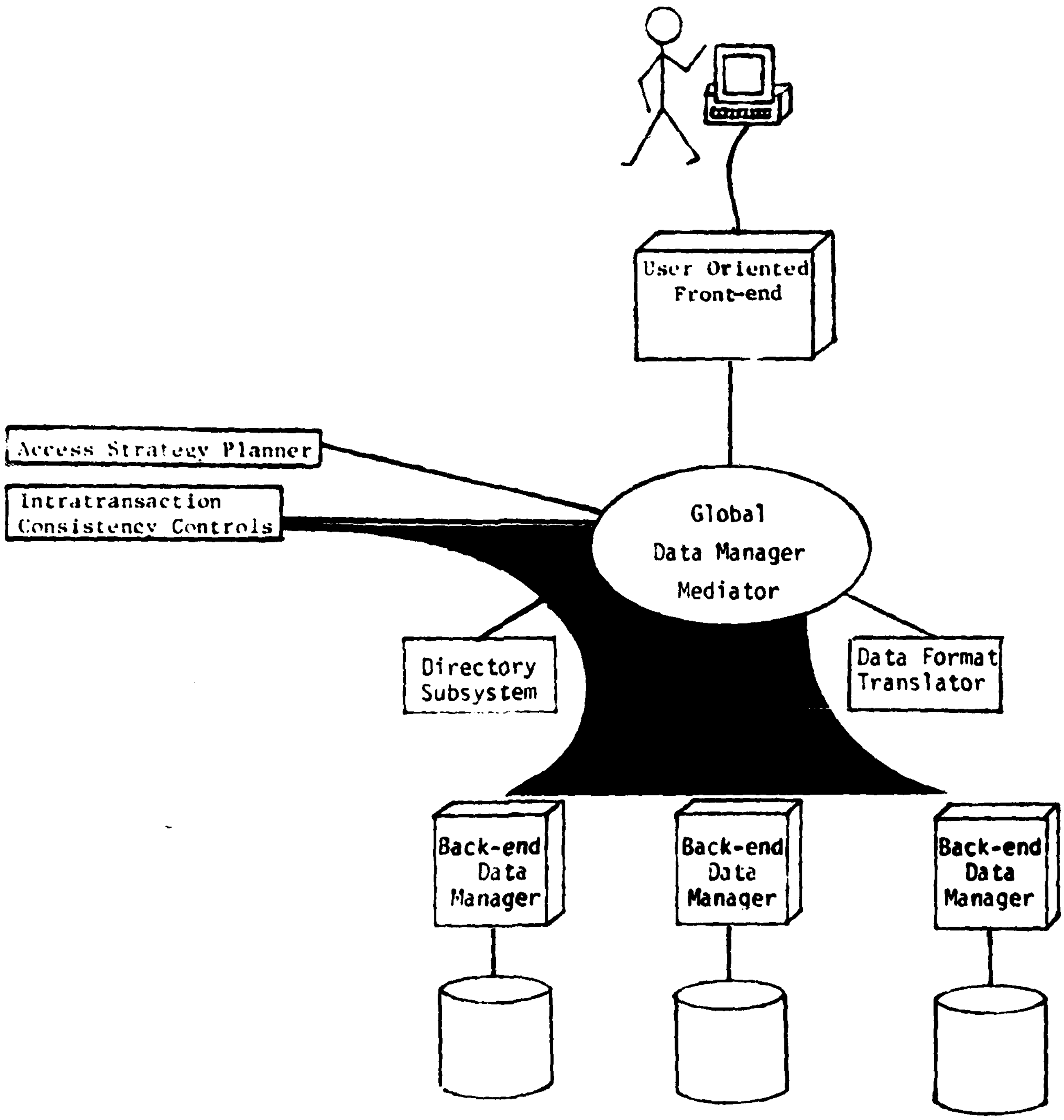


Figure 2-3: Implications of Multiple Back-Ends

clocking),

- a network virtual clock or a method to synchronize the clocks on the individual machines (processor clocking),

- a global scheduler or means of deciding in what (possibly interleaved) order the operations of the different transactions are to be performed,

- an intertransaction consistency analyzer, and

- reliability enhancements and backup mechanisms built into the communications medium.

These requirements are illustrated in Figure 2-4. In the figure several users attached to transaction processors that may be located on one or more nodes of a network access an integrated back-end storage facility which may be distributed across several other nodes. In order to support such access the global data manager illustrated in Figure 2-3 must be enhanced with new functionality to handle potential conflicts among transactions. This functionality is grouped into two main subcomponents: inter-transaction consistency control mechanisms and reliability enhancements - backup mechanisms. Consistency control mechanisms that may be provided include transaction clocking (timestamps), processor clocking (a network virtual clock), a scheduler, and an inter-transaction consistency analyzer. Chapter 4 discusses how these components fit together and gives further details on the rationale for and design tradeoffs among different approaches to implementing them.

2.4 Summary

In this chapter we showed how a simple client-server division of functionality can be used as the architectural basis for some simple systems. We then extended the client-server division of functionality into a more general clients-servers-gdm model for distributed data sharing. In this context we explained why the global data manager (gdm) functional components were needed to support first multiple servers and then multiple clients in a multi-server environment. The next chapter will expand on this by showing how the choice of point of division between client and server can be made based on a hierarchical layering of data abstraction.

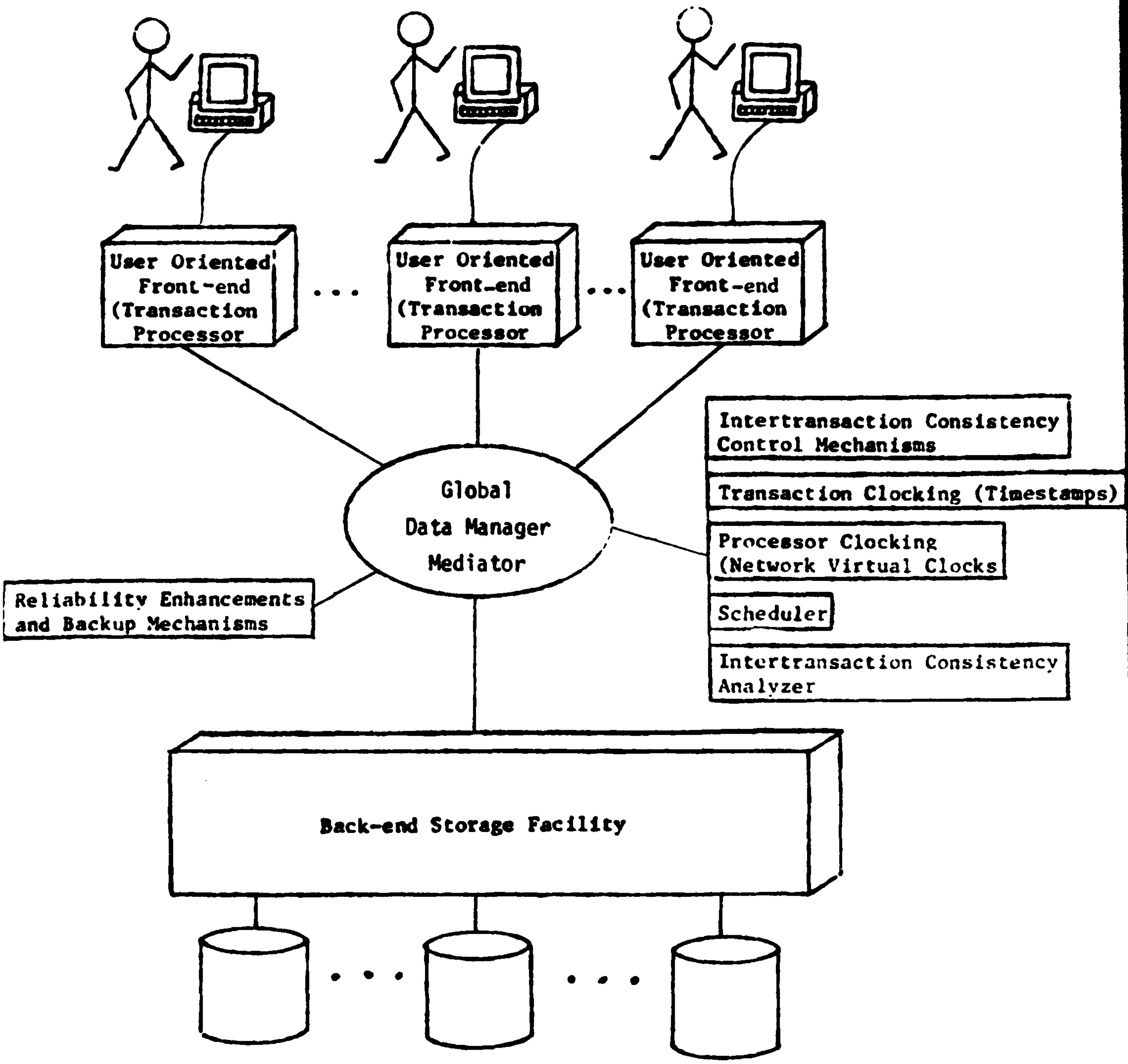


Figure 2-4: Implications of Multiple Front-Ends

3 Decomposition of DBMS Functionality into Abstract Levels

3.1 Overview of Chapter

This chapter gives a paradigm of how a data base management system (DBMS) can be decomposed hierarchicallly. In this decomposition each layer of the DBMS deals with data at a specific level of abstraction. One such choice of levels of abstraction is described in this chapter.

The usefulness of this decomposition from the viewpoint of our formal reference model is that high levels of abstraction can be implemented in a client (abstract) machine, low levels of abstraction in a server (abstract) machine, and a particular level of abstraction chosen as the interface between the two machines. Thus, even though the layering was originally developed for centralized DBMSs (residing in only a single node), its main use in our model of a distributed data sharing system is to provide a mechanism to divide DBMS functionality into front-end (client) and back-end (server) parts. It turns out, as we shall see from the examples presented later in this chapter, that the use of this type of layering is a very general way of modeling the division between client and server machines.

3.2 Proposed Layering for Database and File Management Systems

Conventional data base management systems (on a single computer) can be decomposed using 8 hierarchical levels of abstraction. One such decomposition is shown in Figure 3-1. In this model two levels are devoted to the physical I/O and file system, three to internal DBMS access functions, and three to end user views of the data base contents. Each level uses the primitives of the level below it to implement a semantically richer view of the data, i.e. one that is closer to the end-user view. Hardware or software handles the interface between levels and does any translation or provision of additional functionality that is required.

This model is a variation on the data access portion of DIAM II (the Data Independent Accessing Model version 2). The DIAM model was originally

LEVEL NO.	LEVEL NAME	CONSTRUCTS AND MAPPINGS VISIBLE IN TYPICAL DATA MODELS		CORRESPONDING DICTIONARY/DIRECTORY VIEW
Level 7	END USER COMPOSITE INFORMATION LEVEL	QUERY RELATIONS	COMPOSITE OR GLOBAL OPERATIONS	EXTERNAL SCHEMAS (SUBSCHEMAS)
Level 6	END USER LOGICAL LEVEL	QUERY INTERNAL INTERFACE (ROWS OF RELATIONS)	PROGRAMMER DML INTERFACES	
Level 5	DBMS INFO-LOGICAL LEVEL	CANONICAL FORMS OF DATA		FULL SCHEMA (CONCEPTUAL SCHEMA)
Level 4	DBMS NAVIGATIONAL LEVEL (STORED LOGICAL RECORD) (DIAM STRING LEVEL	LOGICAL POINTERS BETWEEN RECORDS		INTERNAL SCHEMA (PHYSICAL SCHEMA)
Level 3	FIELD REPRESENTATION LEVEL (DIAM ENCODING LEVEL)	BIT REPRESENTATION OF DATA		
Level 2	FILE SYSTEM LEVEL (DIAM PHYSICAL ACCESS LEVEL)	DIAM FRAMES ACCESS METHODS HASH CODING, INDICES		DIRECTORY SYSTEM
Level 1	LOGICAL I/O LEVEL	PHYSICAL BLOCK OR PAGE IN A LINEAR ADDRESS SPACE		
Level 0	PHYSICAL I/O LEVEL	TRACK, SECTOR		VOLUME TABLE OF CONTENTS (VTOC)

Figure 3-1: Hierarchical Decomposition of DBMS Functionality into Abstract Levels

proposed by Michael Senko and his colleagues at IBM [Senko 75] and partially implemented as part of a data base simulator by Martin Marietta [Schneider 75]. A good description of the relevant parts of the DIAM II model is contained in [Sockut 77].

In order to be applicable to as many data base systems as possible the model includes 8 levels corresponding to the hierarchical levels of abstraction in a hypothetical data base management system. This may be a superset of the levels of abstraction provided in any given system in the sense that not all levels need be provided in all systems. For example, the infological level (to be described below) is rarely implemented, but is considered very useful as an aid to conceptual design. In addition, many systems do not provide the end user facilities that correspond to the higher levels of data abstraction. Similarly, lower levels might not be included if an implementation contains some direct execution facility such as associative memory.

In Figure 3-1 each level of data abstraction is divided into three sections that describe:

- the general level descriptor,
- the data objects this would correspond to in some typical data models, and
- the corresponding data dictionary or directory facility.

The <u>end user composite information level</u> (level 7 of the model) contains primitives for processing sets of records grouped together into units such as relations, reports, screens, Codasyl sets, etc. It provides global and macro operations and operations on sets of data objects. It may also support composite objects such as means, sums, standard deviations, and derived objects that are expressable as functions of objects actually stored in the database. The objects can be described in a data dictionary facility, a macro description facility, or in separate screen formatting or report writing utilities.

Level 6, the <u>end user logical data level</u> contains such data objects as the rows of a relation and single logical records. These logical records

may contain "derived data items" that are computed automatically and/or "virtual data items" that are pointed at from within one physical record, but really stored in other physical records. The end user may view a subset of the data base's attributes and relationships and the subset may be viewed as having a certain structure (e.g. a hierarchy, a table or relation, or a directed graph). This end user view may be different from any stored view represented at the navigational level (level 4). Formatting information, logical relationships between data items and records, and access rights are described in the corresponding user subschema. Both level 6 and level 7 contain representations of data (both formats of and linkages between data items) that are visible to end users and application programs. They are for most purposes independent of how data is actually stored in the database.

Level 5, the <u>infological or canonical format level</u> is an intermediate conceptual level that corresponds roughly to the Infological Level in DIAM II [Senko 75]. It provides a global logical view that allows data to be represented in identifier - attribute - value terms that are independent of both the external user (formatted) representation used by application programs and end user facilities and the internal formats and structures used for storing and accessing data on physical media. Data objects are represented in their normal or most general form. The entity-relationship model [Chen 76] and the ANSI SPARC entity attribute model [ANSI/X3/Sparc 75] can be used to represent data at this level. The associated data descriptor is the full schema (or conceptual schema). This level is often implicit rather than explicit in actual implementations.

Level 4, the <u>DBMS navigational level</u> is essentially the same as the DIAM string level. Data objects (records) at this level are accessed in terms of the units they are stored in and the logical relationships between these units. Objects at this level include relations in a relational data base, records and sets in a Codasyl data base, and logical segments and hierarchies in a hierarchical data base. The logical pointers between records are visible at this level allowing a user to "navigate" through the data base by following pointers between records. "Get next record within set", "get logical child", and "find logical parent" are examples of primitives that can be used at this level. The data objects at this level

and the relationships (pointers) between them are described in the physical schema.

Level 3, The field representation level (DIAM Encoding level) defines how data objects (DIAM strings) are represented (encoded) in physical storage. Records and fields are represented at this level as one dimensional bit streams in a linear address space. Character encoding (e.g. ASCII vs. EBCDIC vs. UNIVAC fieldata), integer encoding (e.g. binary vs. BCD vs. packed decimal), field sizes (e.g. 8 vs. 16 vs. 32 bit integers), record sizes (e.g. 6 fields of 4 characters each), and basic storage representation (e.g. "April" vs. "4") are specified at this level. Data encryptation and/or compression and relationship encodings (such as embedded vs. array vs. bit map pointers) are also specified here. Examples of objects at this level are a row of a relation, an IMS logical segment, and a Codasyl record. A record can be a single element of contiguous storage or several such elements chained together via a physical pointer. Examples of the latter would be a linked list and a variable length record extending to an overflow area. Objects at this level are described in the physical schema.

Level 2, the file system or access method level accesses physical segments (records) within a file. Logical fragments, i.e. the parts of a logical record that are stored contiguously, and the access paths to get to them are the basic constructs at this level. Data objects at this level are treated as a contiguous block of n bits, n bytes or n words. Access methods that can be provided include multi-key and regular ISAM, hash coding techniques, b-trees, direct and sequential access, etc. File and record access is supported by a directory system. Access paths may also be described in the physical schema.

Level 1, the virtual address space level, treats secondary storage as a large virtual address space analogous to main memory. This linear address space may be divided into physical pages or blocks and managed with the help of a file management system. The associated directory system keeps track of space allocation, garbage collection, and access rights at the file level. File systems such as Infoplex [Lam 78] and WFS [Swinehart 79] make use of this level. Some operating systems such as IBM's CMS [VM/370 77] and Data General's AOS [AOS 77] use this as an internal

interface to the part of their file system that handles disk space allocation and maintenance.

Level 0, the physical I/O level is the lowest level we mention here and is device dependent. Disk accesses are to track and sector numbers. The associated directory structure is the VTOC (volume table of contents).

3.3 Application to Distributed Systems

In the client / server model described in Chapter 2 one of the levels of Figure 3-1 is chosen as an interface between the client (front-end) and server (back-end) machines. All functionality (levels of abstraction) above this interface is implemented on the client machine (front-end), all functionality below the interface is implemented on the server (back-end).

We call the unit of information transfer at the interface a logical data unit or LDU. The LDU for a given network denotes a canonical form that is the data interface between different machines. Examples of LDUs for some typical systems are listed in Table 3-1. The variation in abstract level of the LDU seems to be an appropriate way to control the relative amounts of functionality to be put in the client (front-end) and server (back-end) parts. Treating the placement of the interface as a parameter in this way makes it easier to see functional similarities between different types of data sharing systems and exposes the underlying commonality of functional requirements in distributed data sharing systems. It may also potentially allow portability of global data manager software between systems that are very different from the point of view of their data model.

Figure 3-2, Figure 3-3, Figure 3-4, and Figure 3-5 show where the systems described in [Mager 80] fit into this model. Xerox and Tandem put the interface at a relatively low level so that their implementations look like distributed file systems (with the possibility of data base functionality being added in the client machine front-end). Cullinane puts the main part of its data base management system in the back-end server with facilities in the client front-end for remote data base access at the DML user subschema level. INGRES and SDD-1 use an intermediate approach that puts most of the data management functionality in the back-end server.

SYSTEM	LOGICAL DATA UNIT (LDU)
XEROX WFS	PHYSICAL PAGES
TANDEM	PHYSICAL RECORDS
SDD-1, INGRES	LOCICAL FRAGMENTS
CULLINANE	DATA BASE RECORDS
	DATA BASE COMPOSITES

Table 3-1: Logical Data Units of Some Typical Systems

LEVEL NO.	LEVEL NAME	CONSTRUCTS AND MAPPINGS VISIBLE IN TYPICAL DATA MODELS	CORRESPONDING DICTIONARY/DIRECT VIEW
Level 7	[redacted]		
Level 6			
Level 5			
Level 4			
Level 3			
Level 2	FILE SYSTEM LEVEL (DIAM PHYSICAL ACCESS LEVEL)	DIAM FRAMES ACCESS METHODS HASH CODING, INDICES	DIRECTORY SYSTEM
Level 1	LOGICAL I/O LEVEL	PHYSICAL BLOCK OR PAGE IN A LINEAR ADDRESS SPACE	*Interface*
Level 0	PHYSICAL I/O LEVEL	TRACK, SECTOR	VOLUME TABLE OF CONTENTS (VTOC)

Figure 3-2: Xerox PARC Client-Server Interface

LEVEL NO.	LEVEL NAME	CONSTRUCTS AND MAPPINGS VISIBLE IN TYPICAL DATA MODELS		CORRESPONDING DICTIONARY/DIRECTORY VIEW	
Level 7	END USER COMPOSITE INFORMATION LEVEL	QUERY RELATIONS	COMPOSITE OR GLOBAL OPERATIONS	EXTERNAL SCHEMAS (SUBSCHEMAS)	
Level 6	END USER LOGICAL LEVEL	QUERY INTERNAL INTERFACE (ROWS OF RELATIONS)	PROGRAMMER DML INTERFACES		
Level 5	DBMS INFO-LOGICAL LEVEL	CANONICAL FORMS OF DATA		FULL SCHEMA (CONCEPTUAL SCHEMA)	Application Program
Level 4	DBMS NAVIGATIONAL LEVEL (STORED LOGICAL RECORD) (DIAM STRING LEVEL	LOGICAL POINTERS BETWEEN RECORDS		INTERNAL SCHEMA (PHYSICAL SCHEMA)	
Level 3	FIELD REPRESENTATION LEVEL (DIAM ENCODING LEVEL)	BIT REPRESENTATION OF DATA			
Level 2	FILE SYSTEM LEVEL (DIAM PHYSICAL ACCESS LEVEL)	DIAM FRAMES ACCESS METHODS HASH CODING, INDICES		*Interface* DIRECTORY SYSTEM	
Level 1	LOGICAL I/O LEVEL	PHYSICAL BLOCK OR PAGE IN A LINEAR ADDRESS SPACE			Encompass
Level 0	PHYSICAL I/O LEVEL	TRACK, SECTOR		VOLUME TABLE OF CONTENTS (VTOC)	

Figure 3-3: Data Interfaces in Tandem Systems

LEVEL NO.	Interface	LEVEL NAME	CONSTRUCTS AND MAPPINGS VISIBLE IN TYPICAL DATA MODELS		CORRESPONDING DICTIONARY/DIRECTORY VIEW
Level 7	*Application Program Interface*	END USER COMPOSITE INFORMATION LEVEL	QUERY RELATIONS	COMPOSITE ON GLOBAL OPERATIONS	EXTERNAL SCHEMAS (SUBSCHEMAS)
Level 6	*Application Program Interface*	END USER LOGICAL LEVEL	QUERY INTERNAL INTERFACE (ROWS OF RELATIONS)	PROGRAMMER DML INTERFACES	*Interface*
Level 5		DBMS INFO-LOGICAL LEVEL	CANONICAL FORMS OF DATA		FULL SCHEMA (CONCEPTUAL SCHEMA)
Level 4	*Back-end Data Manager (IDMS)*	DBMS NAVIGATIONAL LEVEL (STORED LOGICAL RECORD) (DIAM STRING LEVEL	*LOGICAL POINTERS* BETWEEN RECORDS		INTERNAL SCHEMA (PHYSICAL SCHEMA)
Level 3	*Back-end Data Manager (IDMS)*	FIELD REPRESENTATION LEVEL (DIAM ENCODING LEVEL)	BIT REPRESENTATION OF DATA		
Level 2	*Back-end Data Manager (IDMS)*	FILE SYSTEM LEVEL (DIAM PHYSICAL ACCESS LEVEL)	DIAM FRAMES ACCESS METHODS HASH CODING, INDICES		DIRECTORY SYSTEM
Level 1		LOGICAL I/O LEVEL	PHYSICAL BLOCK OR PAGE IN A LINEAR ADDRESS SPACE		
Level 0		PHYSICAL I/O LEVEL	TRACK, SECTOR		VOLUME TABLE OF CONTENTS (VTOC)

Figure 3-4: CDMS Client-Server Interface

	LEVEL NO.	LEVEL NAME	CONSTRUCTS AND MAPPINGS VISIBLE IN TYPICAL DATA MODELS		CORRESPONDING DICTIONARY/DIRECTORY VIEW
Application Transaction (Client) Module	Level 7	END USER COMPOSITE INFORMATION LEVEL	QUERY RELATIONS	COMPOSITE OR GLOBAL OPERATIONS	EXTERNAL SCHEMAS (SUBSCHEMAS)
	Level 6	END USER LOGICAL LEVEL	QUERY INTERNAL INTERFACE (ROWS OF RELATIONS)	PROGRAMMER DML INTERFACES	
	Level 5	DBMS INFO-LOGICAL LEVEL	CANONICAL FORMS OF DATA		FULL SCHEMA (CONCEPTUAL SCHEMA)
	Level 4	DBMS NAVIGATIONAL LEVEL (STORED LOGICAL RECORD) (DIAM STRING LEVEL)	LOGICAL POINTERS BETWEEN RECORDS		INTERNAL SCHEMA (PHYSICAL SCHEMA)
	Level 3	FIELD REPRESENTATION LEVEL (DIAM ENCODING LEVEL)	BIT REPRESENTATION OF DATA		
Data Module	Level 2	FILE SYSTEM LEVEL (DIAM PHYSICAL ACCESS LEVEL)	DIAM FRAMES ACCESS METHODS HASH CODING, INDICES		*Interface* DIRECTORY SYSTEM
	Level 1	LOGICAL I/O LEVEL	PHYSICAL BLOCK OR PAGE IN A LINEAR ADDRESS SPACE		
	Level 0	PHYSICAL I/O LEVEL	TRACK, SECTOR		VOLUME TABLE OF CONTENTS (VTOC)

Figure 3-5: SDD-1, MUFFIN, DIRECT, and Distributed INGRES

However, functionality is provided in the client front-end to translate user requests into a program consisting of accesses to logical fragments. (A logical fragment is the part of a row of a relation that is stored in a physically contiguous location.) This translation is necessary since a single user request can access fragments that are stored on different machines.

3.4 Summary of Chapter

This chapter introduced a new way to decompose access to data in a hierarchical fashion. The approach is based on a hierarchical layering called the Data Independent Accessing Method (DIAM) originally proposed by Michael Senko in the late 1960s and expanded over the next ten years as part of the DIAM II project at IBM. This report uses an extended and modified variation on the DIAM II layering. The modifications were chosen so as to facilitate the application of the model to distributed systems and to make it applicable to distributed file systems as well as to distributed database management. Some examples were given of how typical distributed database and file systems can be described in terms of the model.

4 Distributed Data Sharing Functional Components

The previous chapter described a formal reference model for distributed data sharing in terms of hierarchical levels of abstraction. This chapter describes such a model in terms of the functional components needed to implement a distributed data sharing system. There are two main categories of such functions:

- data accessing functions and
- global data management functions.

The data accessing functions provide traditional data base management services similar to those in centralized systems. The basis for their extension to distributed systems was laid out in the previous chapter in terms of a hierarchical layering and its division into client and server parts.

The global data management functions provide control mechanisms that ensure the integrity of data that is distributed across multiple nodes. Much of the functionality is special to distributed data sharing systems, though there are many parallels and elements of commonality with other distributed resource sharing systems such as distributed operating systems.

This decomposition of data sharing functionality is illustrated in Figure 4-1. In the figure data access functions are divided into the two main categories of chapter 2: client (front-end) functions and server (back-end) functions. Certain functions that don't fit uniquely into either client or server are termed migratory functions and can be implemented in either the client or server part of the system.

Global data manager functions are also divided into three subcategories:

- Intra-transaction consistency control functions that ensure that each transaction leaves the database in a consistent state (provided the database was in a consistent state when the

transaction started),

- inter-transaction consistency control functions that arbitrate and schedule conflicting resource requests from different transactions, and

- reliability enhancements to the network environment that support the continued performance of the data sharing system even in the presence of failures of individual components.

The global data manager serves to coordinate the interaction between clients and servers and ideally makes the fact that the clients and servers are on different machines and, indeed, may be distributed across multiple machines transparent to the data accessing software.

4.1 Data Accessing Functions

The three categories of data accessing functions:

- front-end client functions

- back-end server functions

- migratory functions that can be in either the front-end, the back-end or partially implemented in both

will be described in this section.

4.1.1 Front-end (Client) Functions

The front-end client or application module is generally responsible for

- compiling user requests into a canonical form understandable to the servers

- transforming operations on user visible data items into operations on logical data units

- providing formatting and display functions and other aids to usability.

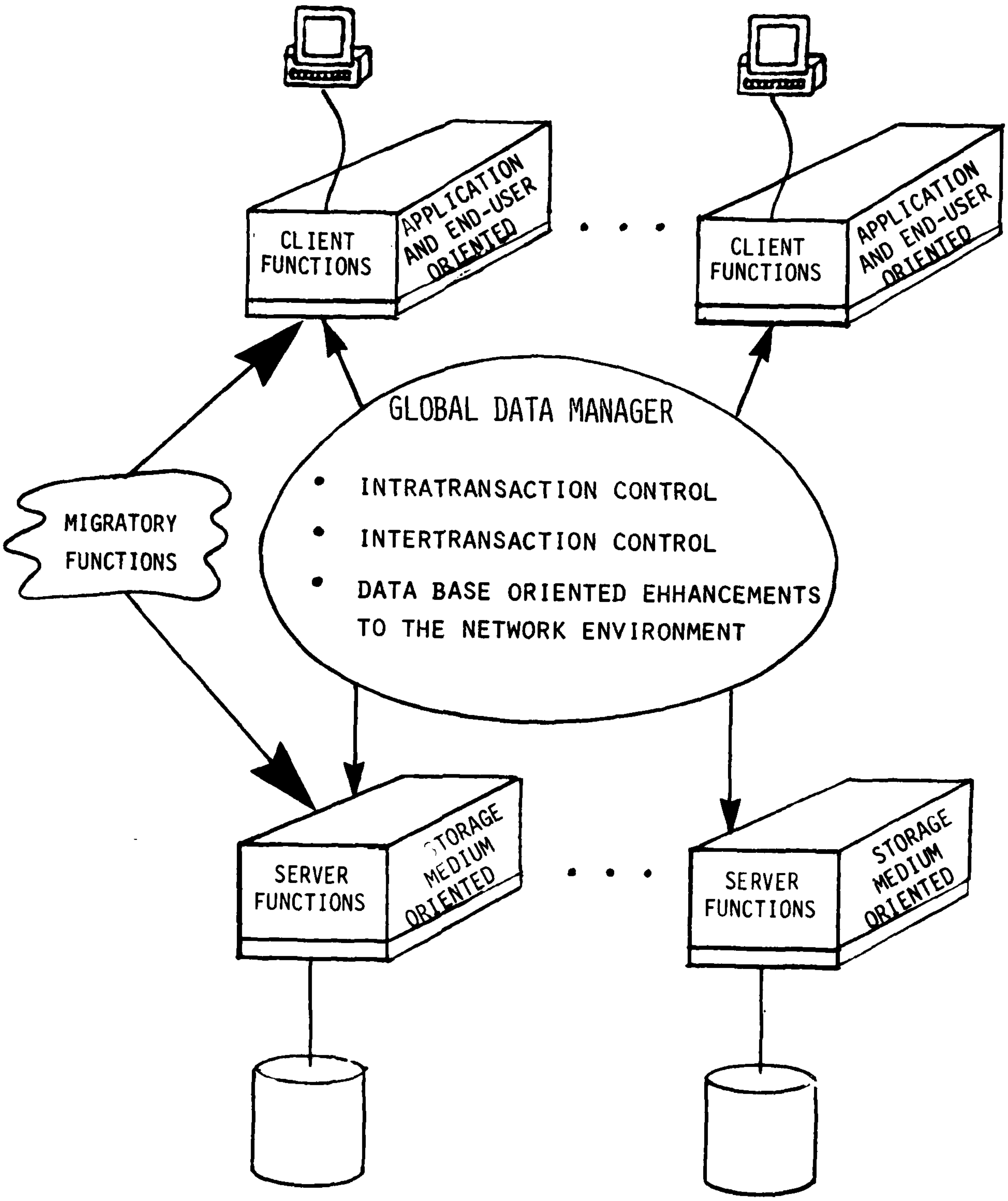


Figure 4-1: Decomposition of Functionality in a Client/Server DBMS

This entails scanning, parsing and doing some semantic interpretation of user commands, interfacing to schemas and subschemas to transform data items between the end user view and the logical data unit view, and then translating the parsed end user request into a sequence of commands to back-end server machines.

The front-end client machine is also responsible for the error handling associated with syntax checking and some semantic checking. The semantic checkijg is to verify compatibility of the user request with data types and integrity information stored in the schema and appropriate subschemas. The client machine thus transforms the user request into a "program" or sequence of messages instructing the back-end servers to perform a sequence of actions on the actual database.

4.1.2 Back-end (Server) Functions

The back-end server machines perform the traditional functions usually associated with database management; that is, they perform actions on their local database in a manner very similar to the way a conventional centralized database manager would handle requests from terminals and other remote users.

These actions include

- data retrieval
- update functions including
    - updates in place of data items within a record
    - insertions
    - deletions
    - changes in indices or access paths
- manipulation functions
- locking of parts of the local database
- certain atomic macro operations such as increment a data item or

counter by one

In general the back-end data managers (servers) execute operations on the database as a sequence of read, manipulate and write actions. (The write and sometimes the manipulate actions can be omitted for strictly query and reporting functions.) Local data items are normally locked from the time of a read until the transaction accessing them has completed. This "locking" may consist of a conventional lock preventing other transactions from taking control of the data item or a more complex "multiple version" scheme. The latter allows other transactions to proceed but preserves old copies of modified data items so that conflicting transactions can be backed out in a consistent manner. Both shared locks (allowing other transactions to read but not to update the data item) and exclusive locks are generally allowed.

For efficiency reasons atomic macro operations allowing reading, manipulating and updating of data items at a single site as an indivisible operation seem attractive. However, choosing just which macro operations to allow is difficult and application dependent because of the need to detect potential conflicts between transactions and compensate for them through scheduling and/or aborting partially completed transactions.

4.1.3 Migratory Functions

The data access functions described above have an essential commonality whatever the abstract level of the interface between client and server. However, certain other functions (mostly concerned with data transformations) can be placed in either the client or server to affect the relative distribution of functionality between these pieces. We term these services <u>migratory functions</u>.

Typical migratory functions include

- operations that transform data objects between higher and lower levels of abstraction/representation
- commands that manipulate or change the value(s) of data objects.

Examples of transformation objects include:

- format and code conversions
- forming derived data items
- replacing pointers with the object pointed to
- data recombinations such as projections and joins

Data manipulation commands include operations such as:

- changes to data items within a logical data unit (such as increasing a value by 10)
- computation of composite data items such as sums, counts, subtotals, etc.

4.2 Global Data Manager

The most critical part of a distributed data sharing system is probably the global data manager that is responsible for coordinating resource allocation and data updates/access on different machines. The global data manager (GDM) can be thought of as containing three parts:

- intra-transaction consistency control (coordination)
- inter-transaction consistency control (mediation)
- database oriented enhancements to the network to reliably support the above requirements. (These enhancements deal mainly with coordinating clocks and time stamps but may also include mechanisms for guaranteeing the reliable transmission of messages in the presence of component failure and some directory functions).

This division is useful for conceptually differentiating between the different tasks of the GDM. In addition, it provides a framework for treating the design of GDM functional modules as separate problems and for partitioning the distributed data management problem into separate localized tasks. In particular, the intra-transaction coordination mechanisms can usually be separated out into separate modules that can be

allocated on a per transaction basis and freed from concern about interaction with other transactions.

4.2.1 Intra-transaction Consistency Control (Coordination)

Intra-transaction coordination functions generally fall into one of three categories:

- access planning
- coordination (monitoring) of distributed query execution
- reliability mechanisms that ensure that either the entire transaction is executed or none of it is.

4.2.1.1 Access Planning

The <u>access planning</u> function generally consists of the following six steps:

1. determining where (on which nodes) copies of data to be accessed are stored. (These copies constitute the transactions <u>readset</u>).

2. determining the most convenient copy of each data item to access (when some data items are stored redundantly). This may involve determining the order of evaluation of boolean expressions and may depend on estimates of the relative frequency of occurrence of different keys.

3. determining which parts of the user request can be executed in parallel. This involves analyzing the dependency relationships in Boolean expressions and other actions (such as following pointer chains). The final determination may be data dependent and involve several alternative execution paths.

4. determining the most appropriate machines to execute the p[illegible]es of the transaction. This will, in general, depend on the locations of data items, the processing power and special facilities of the different nodes (including those needed to support security requirements), the bandwidth of communication links, and node and path utilizations. Resource utilization information can be either dynamic (based on resource monitoring) or static (sysgened into the system and updated offline). Steps (2), (3) and (4) are interdependent and may have to be solved iteratively or as part of a single algorithm.

5. determining what data might be updated by the transaction (the transaction's writeset. Each copy of data items in the writeset must be locked or otherwise flagged to prevent access requests by other transactions that could lead to inconsistencies. The writeset of a transaction may later be refined during execution to release data items that turn out not to be needed by the transaction. (This may occur when the choice of which items to update is determined dynamically during the course of the transaction). Note that if all copies of the data to be updated are flagged (locked) in this stage, it is sufficient for only a single copy to be flagged (locked) in step 1.

6. using the information gathered in steps (1) through (5) to set up a "distributed program" to perform the user request on one or more machines.

4.2.1.2 Run-time Coordination

Coordination of distributed query execution entails initiating pieces of the "distributed program" generated in the access planning stage on appropriate server machines as the input data for each piece becomes available and verifying their successful execution. This involves

- synchronizing the parallel sub-programs running on different machines.
- supervising the transfer of information between subprograms
- refining the transaction's writeset (so as to release locks on data that turn out not to be needed or to unflag data items if the locking is done only when modified data is actually sent)
- monitoring network events (such as machine failures) that might force changes in the transaction execution plan, and
- dynamically handling any errors that occur.

4.2.1.3 Intra-transaction Reliability Mechanisms

Various intra-transaction reliability mechanisms can also be incorporated in the intra-transaction coordination function. The most widely used of these is two-phase commit which can be implemented in a variety of ways [Gray 78], [Rothnie 79], [Stonebraker 77]. The basic principle is that either all database changes made by a transaction should be incorporated into the database, or none should. Guaranteeing this

entails making sure that all nodes have made whatever changes to data they plan to in a separate work area prior to any node transferring modified data back to the database. When it is verified that all data modifications have been made, a commit message is sent to each node to allow its part of the database to be updated.

Various algorithms are used to guarantee that all updates will be incorporated in the database even if some nodes fail during the course of the transaction. In general, the transaction will be completed if the coordinating node stayed up long enough to send a commit message to at least one other node; otherwise the transaction will be aborted and started over from the beginning (or from a predefined checkpoint).

Recovery and verification of status is handled by a dialogue among nodes taking part in the transaction that are still operational.

4.2.2 Inter-transaction Consistency (Mediation)

Inter-transaction consistency control or mediation is probably the most complex and least understood part of distributed data sharing systems. The basic problem is to ensure that the effect of a group of transactions executed in parallel is equivalent to executing the same group of transactions serially (one at a time) in some order. This property is known as <u>serializability</u>.

Serializability is not a problem as long as the transactions do not access common data items (or access them in a read-only manner). Such transactions are said to be <u>mutually transitive</u> in the sense that the order in which they are executed can be reversed or overlapped without affecting the state of the database. However when two concurrent transactions wish to update the same data item, or when one wishs to update a data item that a second transaction needs to read, there is a potential conflict that must be resolved by the mediation mechanism. The inter-transaction concurrency control mechanism resolves this conflict by arbitrarily forcing the conflicting transactions to be serviced in some order. This order may use chronological time stamps, generalized time stamps (or transaction numbers) that take into account explicit priorities, or a more complex token or ticket based mechanism.

4.2.2.1 Synchronization

A basic requirement for the correct operation of an inter-transaction consistency control mechanism is the assignment of a unique global identifier or time stamp or transaction number to each transaction in the system. This time stamp must be unique across all parts of the network which interact with the distributed data sharing system (DDSS) but does not need to have any particular relationship to time in the conventional sense. For example the generalized "time stamp" used by the DDSS may be sensitive to priorities either directly or indirectly or may implicitly give higher priorities to certain nodes by the preallocation of time stamps via a token or ticket mechanism. There are usually significant performance advantages to having the time stamps correspond fairly closely to real time since transactions with later time stamps (higher transaction numbers) must wait for earlier transactions to be completed before they can be finalized (or completed) themselves.

Perhaps the simplest implementation of unique global time stamps is the mechanism proposed by Lamport [Lamport 78]. It entails assigning a unique node number to each node (or transaction generator) in the system and concatenating the appropriate node number with the local clock time to form the global time stamp for transactions generated at that node. This method requires that no more than one transaction be initiated at any given node between any two clock ticks.

It is usually convenient to combine this approach of time stamping for transaction synchronization with a mechanism for processor synchronization so that time stamped messages tend to arrive at data handlers in roughly time stamp order. Such mechanisms usually entail forcing null messages from nodes which would otherwise generate very sparse traffic and forcing nodes to move their clocks ahead whenever they receive a transaction with a later time stamp (transaction number). This last technique might be called the "fastest clock drives the network" approach and seems more viable when the DDSS clock gives the transaction (and indirectly the database version) number rather than an approximation to real time. It tends to keep the clocks or priority ticket allocations at the different nodes of the system roughly in synch.

An alternative to the Lamport approach is to force the transactions into a global serial order via a circulating token or ticket allocation scheme for transaction numbers. Workers at IRIA [LeLann 77] [LeLann 78] [LeBihan 80] have proposed doing this by organizing the nodes generating transactions into a logical virtual ring.

A control token or sequencer circulates around the ring. When a node has control of the token it may take a certain number of tickets that correspond to transaction numbers or time stamps and assign these to transactions that originate at that node. The number of tickets a node should grab is the crux of the algorithm and an important design decision. Possibilities include:

- taking time stamps only for transactions already initiated at that node but still awaiting time stamps, and
- various schemes that involve taking more time stamps or tickets than are immediately needed so that new transactions that are initiated before the token returns can receive a time stamp without delay.

The trick in the later case is to assign "future time stamps", i.e. tickets, with transaction numbers a certain distance into the future leaving a gap of numbers for assignment to transactions that have already started at other nodes. Just how large these gaps should be and how many tickets should be allocated in this way is a crucial performance problem that seems to be very application dependent.

4.2.2.2 Locking Mechanisms

There are two basic approaches to enforcing time stamp order on database accessing:

- locking a data object while it is being updated or read
- maintaining multiple versions of data objects being updated.

Locking an object at a single node is the conventional way of serializing concurrent operations that may conflict. (Note that we talk

about locking data objects rather than data items to make the discussion independent of the granularity of locking chosen.) Two types of locks are normally required:

- exclusive locks for write access
- shared locks for read access. The second type of lock has the principal effect of preventing other transactions from obtaining a write lock on the object.

The generalization to locking objects that are stored redundantly at several nodes involves complex issues and allows a large number of alternative approaches. These include using:

- a centralized lock manager that sets and releases all locks for the entire DDSS. This is the simplest to implement, but not always the most efficient.
- distributed lock managers at each site but with all lock and release requests for any given object routed through a primary site. The copy of the data object at that site is called the primary copy of the object. The primary site lock manager is responsible for controlling the locking and releasing of duplicate copies of objects it controls located at other sites through a message dialogue with the local lock manager at those sites.
- distributed lock managers with symmetric allocation of control. Consistency between locking states at different nodes is guaranteed by the protocol or message exchange between nodes rather than by a module at any one site.
- distributed symmetric lock managers with voting. This is a variation of the previous method that allows locks to be granted/released even if some of the nodes that contain copies of the data object are inaccessible because of either node or network failure. Thomas [Thomas 79] has shown that this can be done consistently as long as a majority of the nodes with copies of the data object participate in the dialogue.

4.2.2.3 Multiple Version Approaches

An alternative to locking data objects being updated is to allow simultaneous access to a data object by multiple transactions but to maintain multiple versions of that data object so that all transactions accessing that object see a consistent version of the database. This approach generally requires attaching a time stamp to each data object in the database (or at least to those "currently" being dealt with). This introduces significant storage overhead in some cases. An optimization proposed by Thomas [Thomas 79] maintains time stamps only for active data objects and purges old time stamps when it is clear that there is only one version of the object currently active in the system (i.e. no updates against that object in progress). Old versions of data objects can be purged from the system when it is determined that there are no longer any transactions in the system that might access them. A newer and possibly more efficient variation on this has been proposed by Takagi [Takagi 79].

Multiple version schemes effectively assign a new version of the data base to each transaction corresponding to the state the database will be in when that transaction is completed. These versions might be conveniently identified by the transaction number. When a transaction wishes to update a data object being accessed by another transaction, a new version of that data object (and implicitly a new version of the database) is created. If no conflicts arise between the updates being performed by the concurrent transactions, both results are kept in the database and older versions are eventually deleted. If a conflict arises then the later transaction (and all transactions dependent on it) are backed out and restarted using new values introduced into the data object by the transaction with the earlier time stamp.

Multiple version approaches seem attractive when a large proportion of database transactions are read-only queries or report requests and/or when predeclarations are used to reserve a larger area of the database than is actually read or modified. Queries and report requests can often be satisfactorily serviced by processing them against a slightly outdated version of the database; this may often be preferable to the degradation in response time that would be caused by waiting for all updates in progress

to quiesce to a consistent state.

Similarly it may sometimes be more efficient to proceed with an update transaction using an older version of some data object on the assumption that any changes being made to that data object by other transactions will not affect the results of the first transaction. This seems likely to be true when the granularity of both objects being time stamped is large relative to the data object actually being updated.

The actual performance tradeoffs between pure locking mechanisms and multiple version approaches are complex, application dependent and dependent on the workloads at the various nodes as well as the characteristics of the database.

There are two principal types of mediation which the GDM must perform between operations of different transactions.

- write-write mediation where 2 different transactions wish to update the same data item,
- read-write mediation where one transaction wishes to access a data item being updated by a second transaction

Read-read mediation involves no conflict but probably forces the GDM to maintain use counts (or use lists) on data items to keep track of how many (and which) transactions are currently accessing them.

It has been argued that different algorithms could be used for the two subproblems, possibly encompassing a combination of pure locking and multiple version approaches. However, it becomes crucial to assure the mutual compatibility of the group of algorithms eventually chosen and the performance analysis of such systems is far from simple. Some of these tradeoffs have been discussed by [Badal 80] and [Bernstein 78].

4.2.2.4 Deadlock Control

Another important part of inter-transaction consistency control is the deadlock control mechanism. Two basic approaches can be used:

- deadlock avoidance
- deadlock detection and resolution, which implies the ability to cancel and back out of a transaction.

There are a number of ways to guarantee that a resource allocation mechanism will be deadlock free, but it is difficult to apply these techniques to data sharing systems. The basic requirement is to allocate all resources (or data objects to be locked) at once or in batches arranged in some hierarchical order. A problem is that it is not always possible to know in advance all data items that will be accessed by a transaction. In particular, the first part of a transaction may determine what data items are to be read or updated in the second part. Preanalysis plus overcommitting of resources (by locking on a larger part of the database than is actually needed) can get around this problem, but usually imply severe performance penalties. These may be at least partially obviated by multiple version consistency schemes. In addition, it is questionable whether there is a useful way to partition a database into a hierarchical order for purposes of locking on data objects in most applications.

Deadlock detection and resolution is an alternative approach which has been successfully used in Distributed Ingres [Stonebraker 78]. Roughly the same basic approaches to division of control can be used for deadlock detection as for consistency control. These include:

- centralized deadlock detention
- an hierarchical form of deadlock detection where deadlocks are tested for in subnetworks with many interactions prior to being tested for in more loosely coupled parts of the network
- distributed deadlock detection algorithms which depend on a dialogue between deadlock detectors at each node to determine conflicts

A wait for graph is sometimes used to indicate which transactions are waiting for each other and thus potentially in conflict.

4.2.3 Fault Tolerance

Intra- and intertransaction control mechanisms need to be supported by network facilities to ensure reliable and timely delivery of messages. The basic requirement is to guarantee correct and in sequence delivery of messages between any two nodes. This is to some extent supported by the message level windowing between socket connections in networks such as ARPANET. Other networks may require special sequencing and flow control facilities be implemented as support functions within the data sharing system itself.

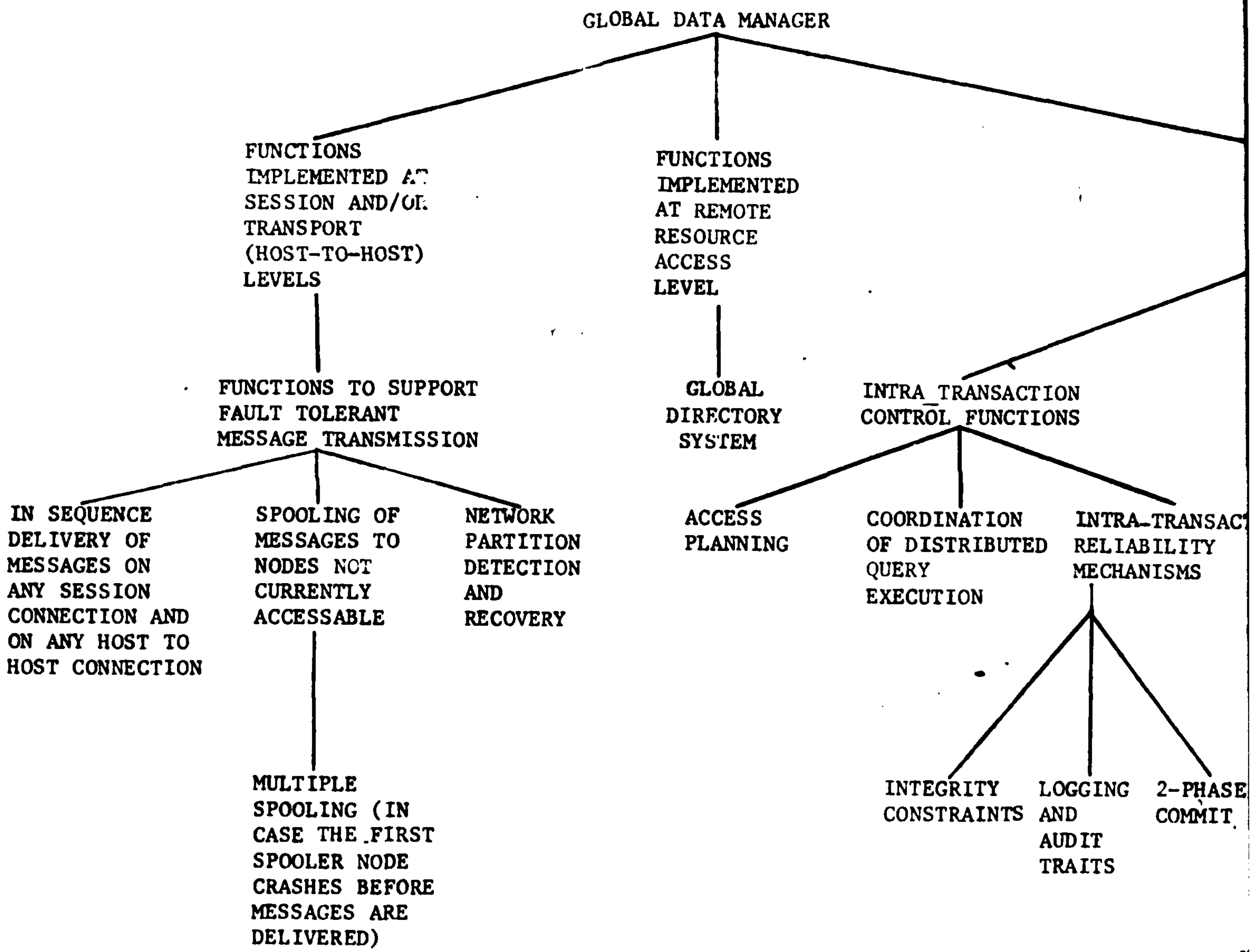
In addition a spooling and forwarding facility is required to ensure that messages sent to a node that is temporarily down or disconnected from the network can be delivered to it in the correct order when it comes up or rejoins the network. Multiple spoolers can be implemented to provide a backup if the node containing a spooler goes down prior to delivering its log of messages.

Various majority consensus type voting schemes have been proposed [Thomas 79] to allow a DDSS to continue to operate when the system is partitioned into two or more parts due to network failure. However there is very little experience with these techniques in practical situations, and systems such as SDD-1 still require manual intervention to deal with the problem.

4.3 Summary

An overview of the pieces of the global data manager and how they are related is shown in figure Figure 4-2.

The question sometimes arises of how to distribute the Global Data Manager functions between client and server parts of a distributed data sharing system. It is conceptually useful to regard the GDM as a separate entity implemented in a distributed fashion. However, for purposes of implementation it seems practical to distribute transaction dependent


GLOBAL DATA MANAGER
FUNCTIONS IMPLEMENTED AT SESSION AND/OR TRANSPORT (HOST-TO-HOST) LEVELS
FUNCTIONS IMPLEMENTED AT REMOTE RESOURCE ACCESS LEVEL
FUNCTIONS TO SUPPORT FAULT TOLERANT MESSAGE TRANSMISSION
GLOBAL DIRECTORY SYSTEM
INTRA_TRANSACTION CONTROL_FUNCTIONS
IN SEQUENCE DELIVERY OF MESSAGES ON ANY SESSION CONNECTION AND ON ANY HOST TO HOST CONNECTION
SPOOLING OF MESSAGES TO NODES NOT CURRENTLY ACCESSABLE
NETWORK PARTITION DETECTION AND RECOVERY
ACCESS PLANNING
COORDINATION OF DISTRIBUTED QUERY EXECUTION
INTRA-TRANSAC
RELIABILITY MECHANISMS
MULTIPLE SPOOLING (IN CASE THE FIRST SPOOLER NODE CRASHES BEFORE MESSAGES ARE DELIVERED)
INTEGRITY CONSTRAINTS
LOGGING AND AUDIT TRAITS
2-PHASE COMMIT

U
T
T

1

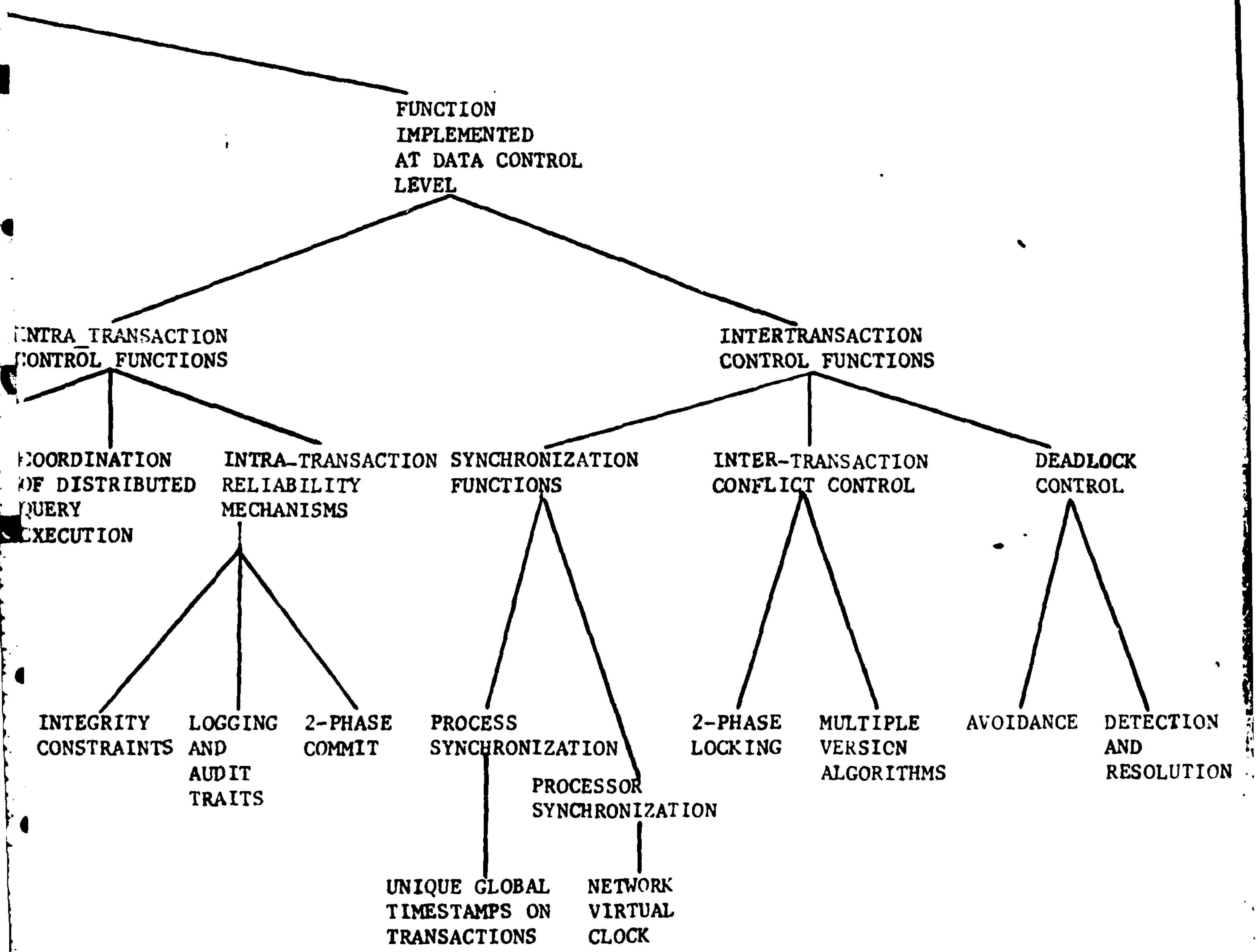

FUNCTION IMPLEMENTED AT DATA CONTROL LEVEL
NTRA_TRANSACTION ONTROL FUNCTIONS
INTERTRANSACTION CONTROL FUNCTIONS
OORDINATION F DISTRIBUTED UERY XECUTION
INTRA-TRANSACTION RELIABILITY MECHANISMS
SYNCHRONIZATION FUNCTIONS
INTER-TRANSACTION CONFLICT CONTROL
DEADLOCK CONTROL
INTEGRITY CONSTRAINTS
LOGGING AND AUDIT TRAITS
2-PHASE COMMIT
PROCESS SYNCHRONIZATION
PROCESSOR SYNCHRONIZATION
2-PHASE LOCKING
MULTIPLE VERSION ALGORITHMS
AVOIDANCE
DETECTION AND RESOLUTION
UNIQUE GLOBAL TIMESTAMPS ON TRANSACTIONS
NETWORK VIRTUAL CLOCK

Figure 4-2: The Subfunctions of the Global Data Manager

functions among the client parts of the system and data copy dependent functions among the servers.

Functions designed to ensure reliable receipt of messages and other network recovery functions probably are required at each node (whether it is a front-end client, back-end server, or combination containing the functionality of both).

5 Conclusion

This report describes a new formal reference model for distributed data sharing that generalizes on conventional reference models for DBMSs (such as Senko's DIAM II model [Senko 75]) in the following ways:

- It extends conventional notions of data modelling to incorporate issues related to sharing data across the nodes of a network. It does this in a manner that treats data modelling issues and consistency - remote access issues as separate dimensions of the task of sharing distributed data.
- It extends the levels of the DIAM II model (which describes DBMS architectural levels) by incorporating two lower levels (for physical block and device access) and one higher level (for composite data types and applications). This is useful for presenting a unified discussion that spans both distributed DBMSs and distributed file systems.

This produces an integrated reference model (for distributed processing) that divides DBMS functionality into client (front-end), server (back-end) and global data manager (distributed data sharing) parts. The division between client and server (front-end and back-end) is kept flexible by making the level of data abstraction for the interface between front and back-end parts a parameter of system design. This allows our reference model to be applicable to a wide assortment of distributed DBMSs and distributed file systems that support vastly different data models.

It is hoped that treating both types of systems in a uniform manner will make it easier to recognize functional similarities between systems and design portable components. This could make it possible to design future data sharing systems starting from a basic template and tailoring the systems for specific needs by appropriate choice of parameters and algorithms. It may also lead to greater interchangeability of functional components as the common nature of functional modules in distributed data sharing systems becomes more evident.

This type of approach can be of use to the Navy in the following ways:

- It provides a framework for analyzing how existing systems (especially systems which contain separate small data bases or files) can be integrated together into distributed or federated systems. This integration can be performed even if the files and databases support different data models and use interfaces at different levels of abstraction.

- It provides a way of developing standard interchangeable components to support specific functions. This can lead to more modular system that are easier to maintain. It may also allow the more rapid introduction of more efficient and more reliable algorithms since the functional units that incorporate them will be treatable as separate modules. In particular the functional division between data sharing and data accessing components of the system may facilitate greater interchangeability of data sharing algorithms among systems which use different data models.

The above two considerations tend to decrease both system cost and the lead time needed for the introduction of new capabilities. In addition distributed systems can be designed to be more reliable and available than single node systems by making use of hardware and data redundancy to support continued system operation in the presence of individual component failures.

Three areas in particular seem to warrant further study as possible extensions of this model. These are:

- integration of this data model with a standard network layering such as that proposed in the ISO Open Systems Architecture.

- the use of this type of model as an aid to the design of global data managers that handle the interfaces between client packages (application programs and end users) and an assortment of back-end (server) data base managers and file systems.

- the use of this model as a conceptual basis for the performance modelling of distributed data management.

6 References

[ANSI/X3/Sparc 75]
ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report. FDT (ACM SIGMOD) , February, 1975.

[AOS 77] Advanced Operating System (AOS) Programmer's Manual 093-000120-01 edition, Data General, 1977.

[Badal 80] Badal, D.Z. The Analysis of the Effects of Concurrency Control on Distributed Data Base System Performance. In Sixth International Conference on Very Large Data Bases. IEEE, October, 1980.

[Banerjee 78] Banerjee,J., Baum,R.I., and Hsiao,D.K. Concepts and Capabilities of a Database Computer. Transactions on Database Systems 3(4), December, 1978.

[Banerjee 79] Banerjee,J., Hsiao,D.K., and Kannan,K. DBC - a Database Computer for Very Large Databases. IEEE Transactions on Computers c-28(6), June, 1979.

[Bernstein 78] Bernstein, Philip, Rothnie, James, Goodman, Nathan, and Papadimitriou, Christos. The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case). IEEE Transactions on Software Engineering SE-4(3), May, 1978.

[Bernstein 79] Bernstein, P.A., Shipman, D.W. and Wong, W.S. Formal Aspects of Serializability in Database Concurrency Control. IEEE Transactions on Software Engineering SE-5(3), May, 1979.

[Berra 79] Berra,P.Bruce and Oliver,Ellen. The Role of Associative Array Processors in Data Base Machine Architecture. Computer , March, 1979.

[Blake 79] Blake,Russ. Tailor: a Simple Model that Works. In Proceedings of the 1979 Conference on Simulation, Measurement and Modeling of Computer Systems. ACMSigmetrics", August, 1979. ACM Sigmetrics Performance Evaluation Review Vol. 8, No.3.

[Canaday 74] Canaday,R.H., Harrison,R.D., Ivie,E.L., Ryder,J.L. and Wehr,L.A. A Back-end Computer for Data Base Management. Communications of the ACM 17(10), October, 1974.

[Chen 76] Chen, Peter P.-S. The Entity-Relationship Model - Toward a Unified View of Data. Transactions on Database Systems , March, 1976.

[Cullinane 79] Cullinane Data Management System (CDMS)

Cullinane Corporation, Wellesley, Mass., 1979.

[Date 75] Date, Christopher J.
An Introduction to Database Systems.
Addison-Wesley, 1975.

[Gray 78] Gray,J.N.
Lecture Notes in Computer Science. Volume 60: Notes on Data Base Operating Systems.
Springer Verlag, 1978, .

[Hsiao 79] Hsiao, David.
Collected Readings on a Database Computer (DBC).
Technical Report, Ohio State University, 1979.

[Israel 78] Israel, Jay E., Mitchell, James G., and Sturgis, Howard E.
Separating Data from Function in a Distributed File System.
Technical Report CSL-78-5, XEROX PARC, September, 1978.

[Lam 78] Lam,Chat-Yu and Madnick,Stuart.
INFOPLEX Data Base Computer Architecture - Concepts and Directions.
Technical Report, MIT Sloan School Center for Information Systems Research, 1978.

[Lamport 78] Lamport, Leslie.
Time, clocks and ordering of events in a Distributed System.
Communications of the ACM 21(7), July, 1978.

[LeBihan 80] Le Bihan, J.; Esculier, C.; Le Lann, G.; and Treille, L.
Sirius-Delta: a Distributed Data Base Management System Prototype.
North Holland, 1980, .

[LeLann 77] Le Lann, Gerald.
Introduction a l'analyse des systemes multireferentiels.
PhD thesis, Universite de Rennes, May, 1977.
Document SIRIUS CTR.I.004.

[LeLann 78] Le Lann, Gerald.
Algorithms for Distributed Data-sharing Systems which Use Tickets.
In Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks. Lawrence Berkeley Laboratory, University of California, Berkeley, 1978.

[Madnick 75] Madnick, Stuart E.
INFOPLEX - Hierarchical Decomposition of a Large Information Management System using a Microcomputer Complex.
In National Computer Conference. AFIPS, June, 1975.

[Madnick 79] Madnick, Stuart E.
INFOPLEX - Data Base Computer Architecture.
In Compcon Spring 79. IEEE Computer Society, March, 1979.

[Mager 80] Mager, Peter S. and Goldberg, Robert P.
A Survey of Some Approaches to Distributed Data Base and Distributed File System Architecture.
Technical Report TR-80-001, BGS Systems,Inc., Waltham, Mass., January, 1980.

[Martin 77] Martin, James.
Computer Data-Base Organization.
Prentice Hall, Englewood Cliffs, NJ, 1977.
2nd edition.

[Paxton 79] Paxton, William H.

A Client-Based Transaction System to Maintain Data Integrity.
In Proceedings of the Seventh Symposium on Operating System Principles, Pacific Grove, California. ACM (SIGOPS), December, 1979.

[Rothnie 77] Rothnie, James B. and Goodman, Nathan.
A Survey of Research and Development in Distributed Database Management.
In Proceedings of the Third International Conference on Very Large Data Bases. IEEE, 1977.

[Rothnie 79] Rothnie, James B., Jr., Bernstein, Philip A., Fox, Stephen A., Goodman, Nathan, Hammer, Michael M., Landers, Terry A., Reeve, Christopher, L. Shipman, David W., and Wong, Eugene.
SDD-1: A system for Distributed Databases.
Technical Report CCA-02-79, CCA, August, 1979.

[Schneider 75] Schneider, L.S. and Spath, C.R.
Quantitative Data Description.
Technical Report, Martin Marietta Corporation, 1975.

[Senko 75] Senko, Michael.
Specification of Stored Data Structures and Desired Output Results in DIAM II with FORAL.
In Proceedings of the First International Conference on Very Large Databases. ACM, September, 1975.

[Sockut 77] Sockut, Gary.
Data Base Performance under Concurrent Reorganization and Usage.
PhD thesis, Harvard University, July, 1977.

[Stonebraker 77]
Stonebraker,Michael.
A Distributed Data Base Version of INGRES.
In Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks. Lawrence Berkeley Laboratory, University of California , Berkeley, August, 1977.

[Stonebraker 78]
Stonebraker, Michael.
Concurrency Control and Consistency of Multiple Copies of Data in Distributed Ingress.
In Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks. Lawrence Berkeley Laboratory, University of California, Berkeley, 1978.
reprinted in IEEE Transactions on Software Engineering, Vol. SE-5, No.3, May 1979.

[Stonebraker 79]
Stonebraker, Michael.
MUFFIN: A Distributed Data Base Machine.
In Proceedings of the First International Conference on Distributed Computing Systems, Huntsville, Alabama. IEEE Computer Society, October, 1979.

[Swinehart 79] Swinehart,Daniel; McDaniel,Gene; and Boggs,David.
WFS: A Simple Shared File System for a Distributed Environment.
In Proceedings of the Seventh Symposium on Operating System

Principles, Pacific Grove, California. ACM (SIGOPS), December, 1979.

[Takagi 79] Takagi,Akihiro.
Concurrent and Reliable Updates of Distributed Databases.
Technical Report TM-144, MIT LCS, November, 1979.

[Thomas 79] Thomas, R. H.
A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases.
Transactions on Database Systems 4(2), June, 1979.

[VM/370 77] IBM Virtual Machine Facility/370: System Programmer's Guide
GC20-1807-6 edition, IBM, 1977.

[Watson 80] Watson, Richard W.
Network Architecture Design for Back-End Storage Networks.
Computer (2), February, 1980.